CTSC2010珠宝商 新解

南京外国语学校　许昊然

Contents

1 题目大意 **2**

2 预备知识 **2**

2.1 后缀自动机 2

2.2 后缀树 2

3 算法讨论 **2**

3.1 朴素做法A 2

3.2 朴素做法B 3

3.3 朴素做法B的改进 3

3.4 满分做法——改进后的朴素算法B与朴素算法A相结合 3

4 Special Thanks **4**

1 题目大意

给定一棵N个结点的树，树的每个结点上都有一个字符。定义$Path(X,Y)$等于从结点X到结点Y的最短路径所经过的结点上的字符顺次连接起来形成的字符串。给定长度为M的母串S，定义$Occur(X,Y)$为字符串$Path(X,Y)$在母串S中出现的次数。求$\sum_{1\le X,Y\le N} Occur(X,Y)$.

$N,M\le 10^5$，时间限制18秒。

2 预备知识

限于篇幅原因，这部分内容只是为了告诉读者“有这么一个算法可以高效解决某个问题”。 如果想知道具体的“这个算法的原理是什么”，请参考其他资料。

2.1 后缀自动机

某个串的后缀自动机是一个能且仅能识别该串所有后缀的自动机。存在一种$O(N)$的算法构建后缀自动机。建好某母串S的后缀自动机后，可以在$O(Len(T))$的时间内查询某个串T在母串S中的出现次数，并且，在串T末尾新增一个字符并获得新的出现次数只需要$O(1)$时间。

2.2 后缀树

某个串的后缀树是该串的所有后缀组成的trie树。 利用压缩路径表示，存在一种$O(N)$的算法构建后缀树。

3 算法讨论

这道题的官方做法时间复杂度是$O((N+M)\sqrt{N}\log N)$，且十分复杂。 这里我们将给出一种时间复杂度优于官方做法，常数小于官方做法，且更加简洁优美的新做法。

我们不妨先讨论一些朴素的做法。

3.1 朴素做法A

由预备知识，我们可以发现，建立了母串S的后缀自动机后，只需从某个根结点出发DFS一遍，即可$O(N)$获得从这个根结点出发到其子树内各个孩子所构成串在母串中出现的次数。如果我们暴力枚举所有根结点并进行DFS，那么就可以在$O(N^2)$的复杂度内解决本问题。

3.2 朴素做法B

任意两个结点X,Y都必然有一个最近公共祖先。我们考虑所有最近公共祖先是某个结点Z的结点对(X,Y)的$Occur$次数之和。

显然，$Path(X,Y) = Path(X,Z) + Path(Z,Y)$. 考虑结点$Z$对应的字符在母串$S$中匹配的位置。那么，$Path(Z,Y)$显然在母串$S$的该位置匹配，$Path(Z,X)$必定在母串$S$的逆序串的该位置对应位置匹配。 因此，我们如果能统计出母串S的各个位置能匹配多少个$Path(Z,Y)$，以及母串S的逆序串S_{rev}在各个位置能匹配多少个$Path(Z,X)$，那么只需把对应位置的匹配数目相乘，然后求和即是答案。

我们不妨定义$F(Z) = \sum_{X\in Z\text{的子树}} \sum_{Y\in Z\text{的子树}} Occur(Path(X,Z) + Path(Z,Y))$，那么我们所求的以$Z$为根的子树的答案就是$Ans(Z) = F(Z) - \sum_{X\in Z\text{的直接孩子}} F(X)$，也就是所有链对的出现次数和减去最近公共祖先不在$Z$的所有链对的出现次数和。

函数F的计算可以使用我们刚才讨论的方法。 我们建立母串S的后缀树和母串S的逆序串S_{rev}的后缀树，然后从Z结点出发DFS一遍，记录下各个结点在两棵后缀树上对应到达的位置$O(N)$。 然后对两棵后缀树均DFS一遍，把标记推下去到叶子结点（对应着匹配的后缀在串中位置）$O(M)$，从而得到串中的每个位置开始各能匹配多少个Z出发的串，然后把对应位置的匹配数目相乘并求和，即可求出$F(Z)$的值。

要减掉的那部分看似要计算$Deg(Z)$次F函数，但实际上所有结点的孩子的个数之和显然是$N-1$，因此实际会被均摊掉而不影响复杂度。这个做法的复杂度是$O(N+M)$每个Z结点。如果我们直接枚举所有的Z结点并求出$Ans(Z)$的和，那么总复杂度是$O(N^2+MN)$的。

3.3 朴素做法B的改进

实际上，我们发现，朴素做法B的总复杂度可以得到小小的优化。 做法B中的N^2准确的说，应该是各个子树大小的和。 而这是一棵无根树，因此，我们考虑用点分治对这棵树进行分治。 选取树的重心作为根结点进行分治的话，不会影响上面的计算，但我们可以保证我们分治的各个子树的大小的和是$O(N\log N)$级别的，因此总复杂度被优化到了$O(N\log N + MN)$。

3.4 满分做法——改进后的朴素算法B与朴素算法A相结合

有了上面点分治的改进后，我们发现，时间复杂度的瓶颈已经变成了每次DFS后缀树的$O(M)$。 而这个复杂度难以优化。我们发现，计算$Ans(Z)$的复杂度不管Z的子树有多大都始终是$O(M)$，这对于较小的子树显然很浪费。 而朴素算法A的复杂度正好是与M无关的，始终是子树大小的平方。于是我们产生了一个想法，如果分治得到的子树足够小了，就不妨把任务交给朴素算法A来做，这样显然比用朴素算法B划算很多。

我们不妨假设当子树大小不超过某个值S时我们将使用朴素算法A。因为我们选取重心作为根结点进行点分治，所以我们可以保证，每次分治得到的最大子树大小不会超过原子树大小的一半。我们不妨假设我们连续进行了K轮分治，那么最大的子树大小不会超过$\frac{N}{2^K}$。 因此，当分治得到的最大子树大小不超过S时，我们最多进行了$\lfloor \log \frac{N}{S} \rfloor$轮分治，总共分治次数是$2^{\lfloor \log \frac{N}{S} \rfloor} = O(\frac{N}{S})$次。

因此我们得到了一个重要结论： 只需分治$O(\frac{N}{S})$次，就可保证分治出的最大的子树的大小是$O(S)$级别的，而这些子树显然不超过$O(\frac{N}{S})$个，因此我们使用朴素算法A解决这些子树的时间复杂度不超过$O(S^2) * O(\frac{N}{S}) = O(NS)$。 而每次分治需要求一次$Ans$函数来计算跨根结点的答案，复杂度不超过$O(\frac{N}{S}) * O(M) + O(N \log N) = O(\frac{NM}{S} + N \log N)$。 为了使总复杂度最小，我们应当选择$S = \sqrt{M}$，此时总复杂度不超过$O(N\sqrt{M}) + O(M\sqrt{M}) = O((N+M)\sqrt{M})$，足以通过全部数据。

这个算法比官方题解的复杂度少一个log，实际速度是官方做法的2倍以上，且思路较自然，实现也简单很多。

4 Special Thanks

- 感谢中国计算机学会提供了这个交流的平台。
- 感谢龙浩民同学在LaTeX方面给予我的大量帮助，让我能用LaTeX写出这篇题解。